In [1]:

```python
#PS_20174392719_1491204439457_log.csv
# For Data Analysis
import pandas as pd
import numpy as np

# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```python
Fraud_D = pd.read_csv('PS_20174392719_1491204439457_log.csv')

# Remove the last column
Fraud_D = Fraud_D.iloc[:, :-1]
```

In [3]:

```python
Fraud_D.columns= ["step", "type", "amount", "customer_starting_transaction", "bal_before_transaction",
            "bal_after_transaction", "recipient_of_transaction", "bal_of_recepient_before_transaction", "bal_of_receipient_
```

In [4]:

```python
# View data (to give you first five rows)
Fraud_D.head()
```

Out[4]:

	step	type	amount	customer_starting_transaction	bal_before_transaction	bal_after_transaction	recipient_of_transaction	bal_of_recepient_before_trar
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	

In [5]:

```python
# View data (to give you last five rows)
Fraud_D.tail()

```

Out[5]:

	step	type	amount	customer_starting_transaction	bal_before_transaction	bal_after_transaction	recipient_of_transaction	bal_of_recepient_be
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.0	C776919290	
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.0	C1881841831	
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.0	C1365125890	
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.0	C2080388513	
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.0	C873221189	

In [6]:

```python
#Data Verification

Fraud_D.info()

```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 10 columns):
 #   Column                               Dtype  
---  ------                               -----  
 0   step                                 int64  
 1   type                                 object 
 2   amount                               float64
 3   customer_starting_transaction        object 
 4   bal_before_transaction               float64
 5   bal_after_transaction                float64
 6   recipient_of_transaction             object 
 7   bal_of_recepient_before_transaction  float64
 8   bal_of_receipient_after_transaction  float64
 9   fraud_transaction                    int64  
dtypes: float64(5), int64(2), object(3)
memory usage: 485.4+ MB
```

In [7]:
```
# statistical analysis of the data

Fraud_D.describe()
```
Out[7]:

	step	amount	bal_before_transaction	bal_after_transaction	bal_of_recepient_before_transaction	bal_of_receipient_after_transaction	fraud_t
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.3
mean	2.433972e+02	1.798619e+05	8.338831e+05	8.551137e+05	1.100702e+06	1.224996e+06	1.
std	1.423320e+02	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	3.674129e+06	3.
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.0
25%	1.560000e+02	1.338957e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.0
50%	2.390000e+02	7.487194e+04	1.420800e+04	0.000000e+00	1.327057e+05	2.146614e+05	0.0
75%	3.350000e+02	2.087215e+05	1.073152e+05	1.442584e+05	9.430367e+05	1.111909e+06	0.0
max	7.430000e+02	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	3.561793e+08	1.0

In [8]:
```
Fraud_D.describe().astype(int)
```
Out[8]:

	step	amount	bal_before_transaction	bal_after_transaction	bal_of_recepient_before_transaction	bal_of_receipient_after_transaction	fraud_transactio
count	6362620	6362620	6362620	6362620	6362620	6362620	636262
mean	243	179861	833883	855113	1100701	1224996	
std	142	603858	2888242	2924048	3399180	3674128	
min	1	0	0	0	0	0	
25%	156	13389	0	0	0	0	
50%	239	74871	14208	0	132705	214661	
75%	335	208721	107315	144258	943036	1111909	
max	743	92445516	59585040	49585040	356015889	356179278	

In [9]:
```
#Missing values

Fraud_D.isnull()
```
Out[9]:

	step	type	amount	customer_starting_transaction	bal_before_transaction	bal_after_transaction	recipient_of_transaction	bal_of_recepient_before_tra
0	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	
6362615	False	False	False	False	False	False	False	
6362616	False	False	False	False	False	False	False	
6362617	False	False	False	False	False	False	False	
6362618	False	False	False	False	False	False	False	
6362619	False	False	False	False	False	False	False	

6362620 rows × 10 columns

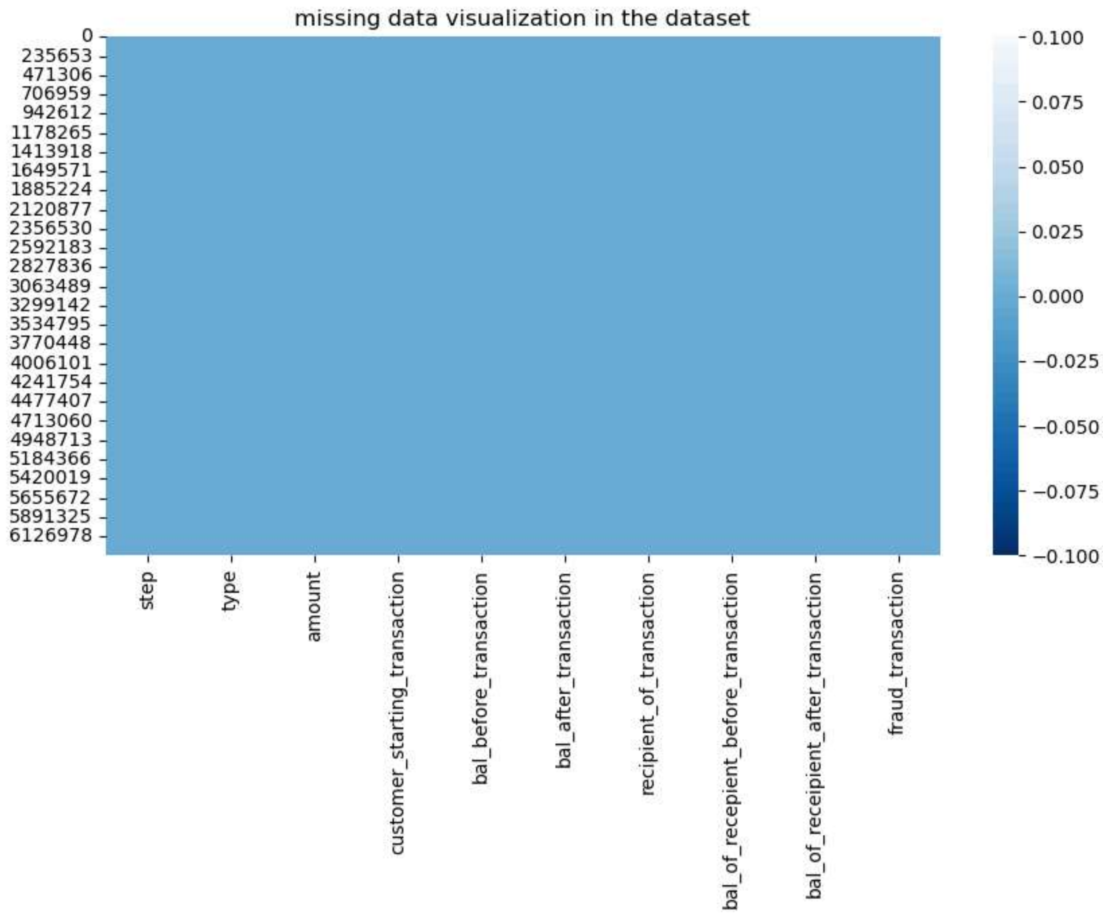
In [10]:
```
Fraud_D.isnull().sum()
```
Out[10]:
```
step                                   0
type                                   0
amount                                 0
customer_starting_transaction          0
bal_before_transaction                 0
bal_after_transaction                  0
recipient_of_transaction               0
bal_of_recepient_before_transaction    0
bal_of_receipient_after_transaction    0
fraud_transaction                      0
dtype: int64
```

In [11]:

```
# To visualize the missing values

plt.figure(figsize = (10,5))
plt.title ("missing data visualization in the dataset")
sns.heatmap(Fraud_D.isnull(), cbar =True, cmap= "Blues_r")
```

Out[11]:

```
<Axes: title={'center': 'missing data visualization in the dataset'}>
```



In [12]:

```
#check shape of the entire dataframe using .shape attribute
Fraud_D.shape
```

Out[12]:

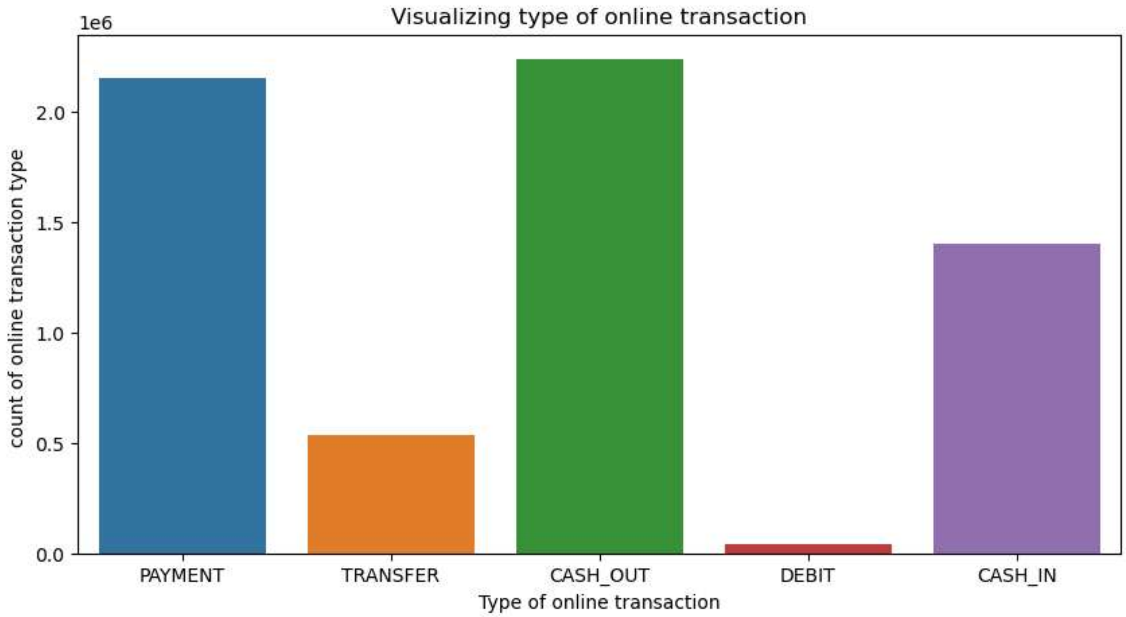
```
(6362620, 10)
```

In [13]:

```
# We have 6362620 rows and 10 columns in the dataset
# EXPLORATORY DATA ANALYSIS
# Univariate Analysis

# Bivariate Analysis

# Multivariate Analysis

# Correlation
```

In [14]:
```
# Univariate Analysis
#visualize type of online transaction
plt.figure(figsize=(10,5))
sns.countplot (x="type", data= Fraud_D)
plt.title ("Visualizing type of online transaction")
plt.xlabel("Type of online transaction")
plt.ylabel("count of online transaction type ")
```

Out[14]:
```
Text(0, 0.5, 'count of online transaction type ')
```



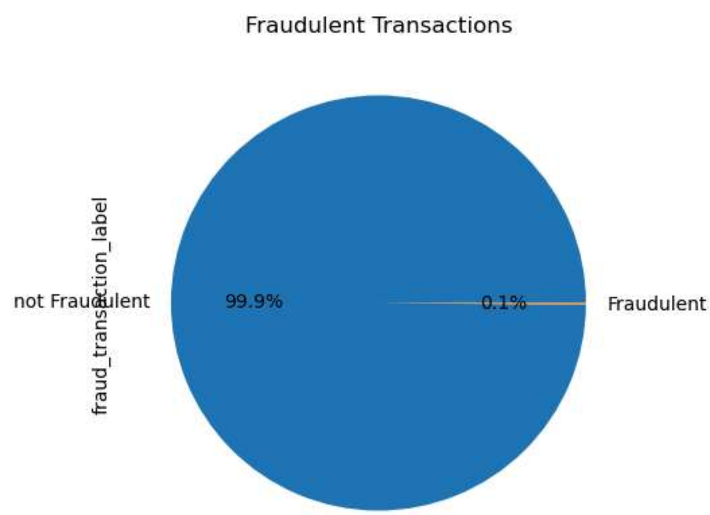
In [15]:
```
# From the chart, it is seen that cash_out and payment is the most common type of online transaction that customers use
```

In [16]:

```python
# create a function that properly labels isFraud

def Fraud (x):
    if x ==1:
        return "Fraudulent"
    else:
        return "not Fraudulent"

# create a new column
Fraud_D["fraud_transaction_label"] = Fraud_D["fraud_transaction"].apply(Fraud)


# create visualization
plt.figure(figsize = (10,5))
plt.title ("Fraudulent Transactions")
Fraud_D.fraud_transaction_label.value_counts().plot.pie(autopct='%1.1f%%')
```

Out[16]:

```
<Axes: title={'center': 'Fraudulent Transactions'}, ylabel='fraud_transaction_label'>
```



In [17]:

```python
# From this chart, its shows that most of the online transactions customers does is not fraudulent. Also the dataset is not
```

In [18]:

```python
Fraud_D.fraud_transaction_label.value_counts()
```

Out[18]:

```
not Fraudulent    6354407
Fraudulent           8213
Name: fraud_transaction_label, dtype: int64
```

In [19]:

```python
8213/6354407*100
```

Out[19]:

```
0.129248881917699
```

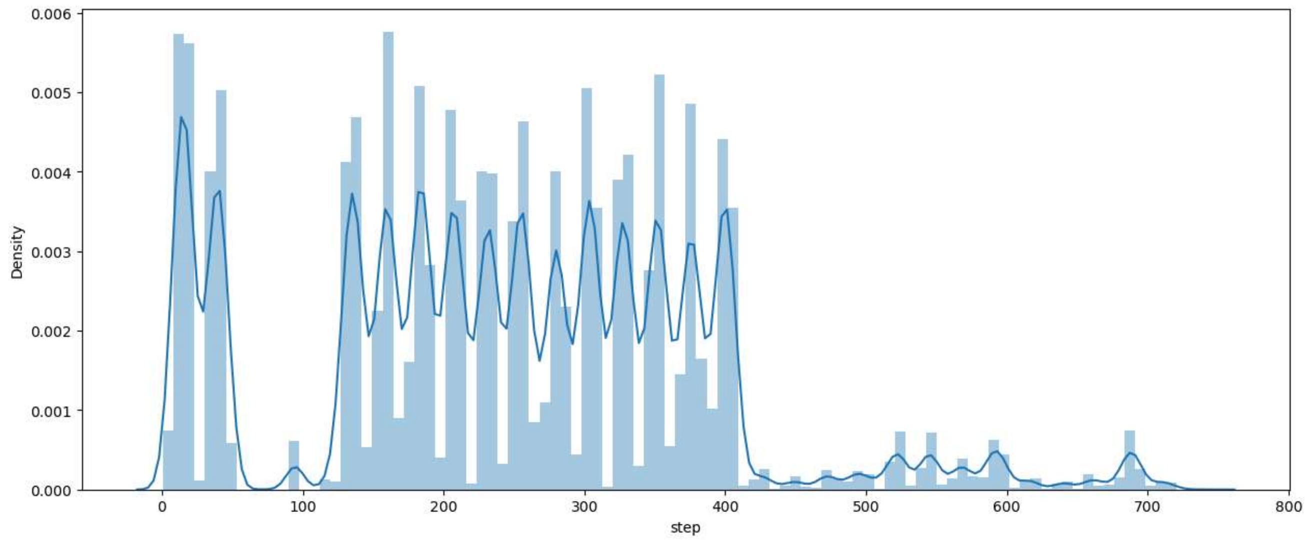
In [20]:

```python
# 8,213 transactions have been tagged as fraudulent in the dataset, which is approximately 13% of the total number of trans
```

In [21]:

```python
#To disable warnings
import warnings
warnings.filterwarnings("ignore")

# Visualization for step column

plt.figure(figsize=(15,6))
sns.distplot(Fraud_D['step'],bins=100)
```

Out[21]: <Axes: xlabel='step', ylabel='Density'>



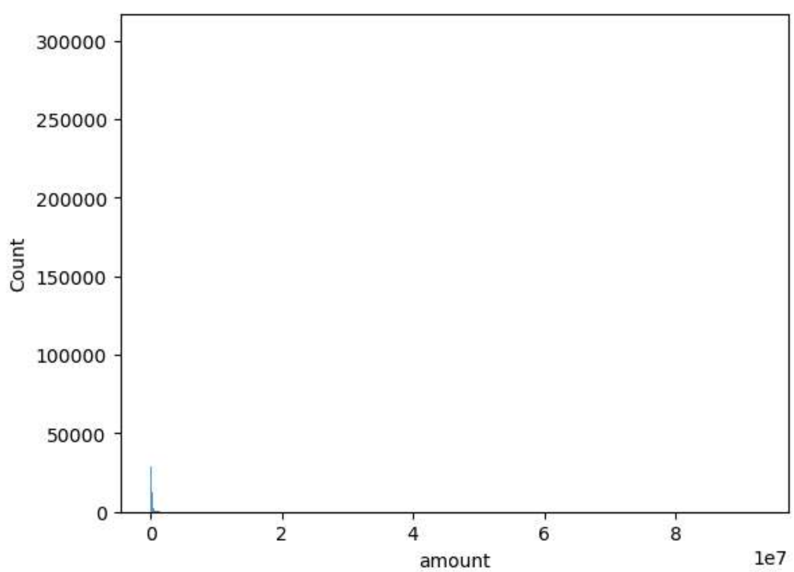
In [22]:

```python
# The above graph indicates the distribution of the step column
```

In [23]:

```python

# Visualization for amount column

sns.histplot(x= "amount", data =Fraud_D)
```

Out[23]: <Axes: xlabel='amount', ylabel='Count'>



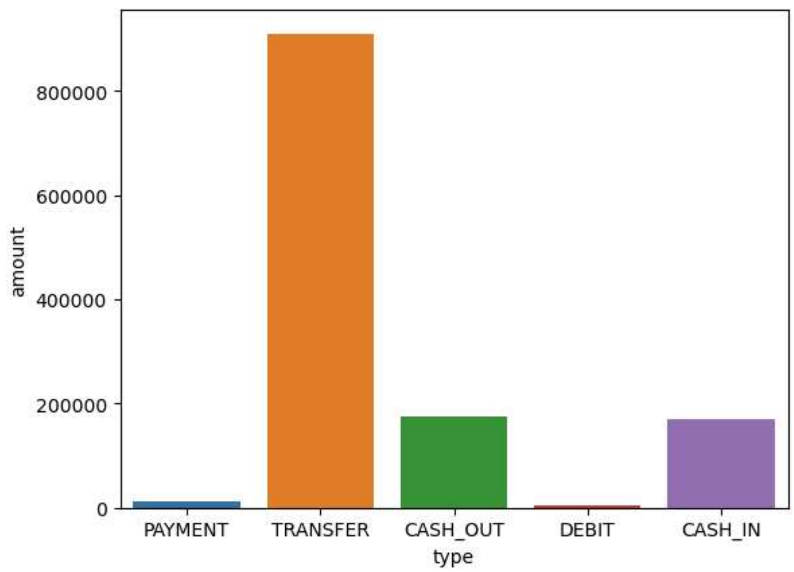
In [24]:
```
Fraud_D.head()
```
Out[24]:

	step	type	amount	customer_starting_transaction	bal_before_transaction	bal_after_transaction	recipient_of_transaction	bal_of_recepient_before_trar
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	

In [25]:
```
Fraud_D.tail()
```
Out[25]:

	step	type	amount	customer_starting_transaction	bal_before_transaction	bal_after_transaction	recipient_of_transaction	bal_of_recepient_b
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.0	C776919290	
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.0	C1881841831	
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.0	C1365125890	
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.0	C2080388513	
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.0	C873221189	

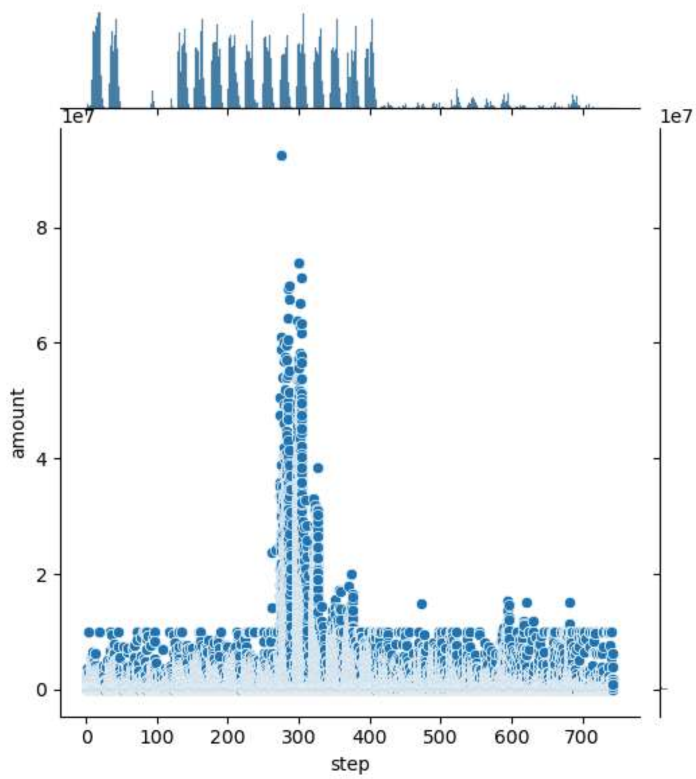
In [30]:
```
# Bivariate Analysis

sns.barplot(x='type',y='amount',data=Fraud_D,ci=None)
```
Out[30]: <Axes: xlabel='type', ylabel='amount'>



In [31]:
```

# In this chart, 'transfer' type has the maximum amount of money being transfered from customers to the recipient. Although
```

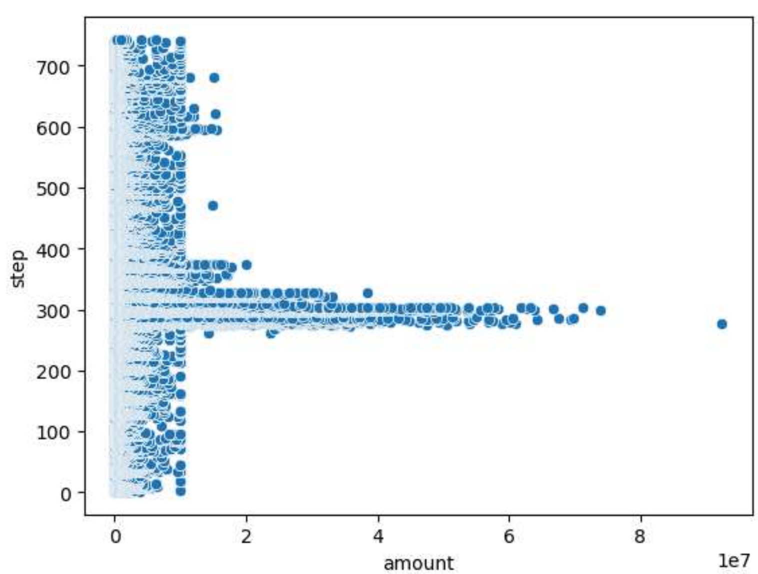
In [34]:
```
# Visualization between step and amount

sns.jointplot(x='step',y='amount',data=Fraud_D)
```

Out[34]: <seaborn.axisgrid.JointGrid at 0x1abb2d5c550>



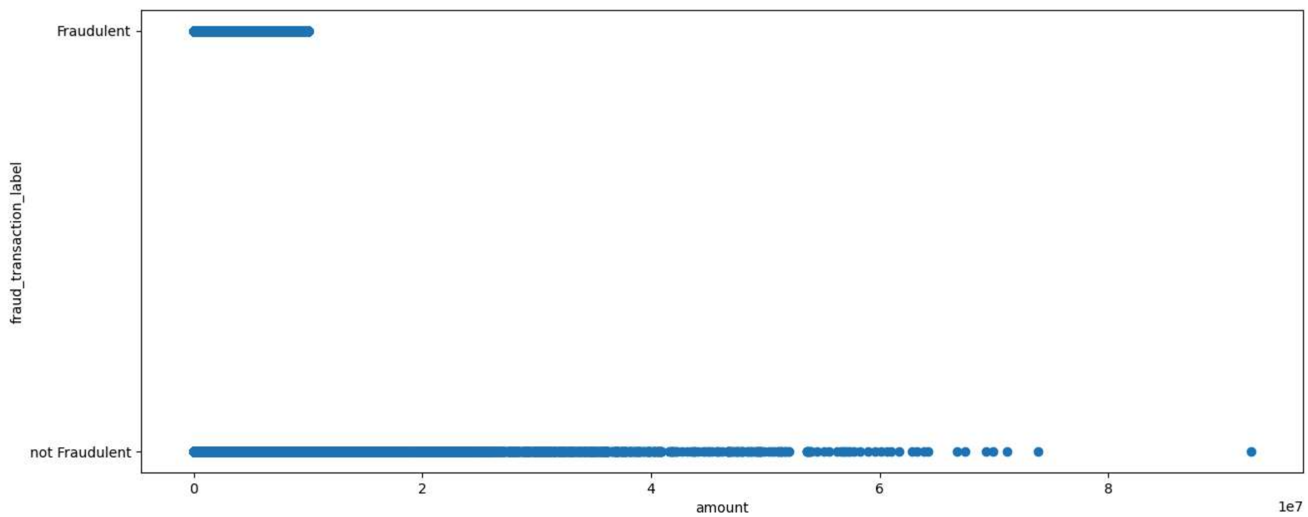
In [35]:
```
sns.scatterplot(x=Fraud_D["amount"], y=Fraud_D["step"])
```

Out[35]: <Axes: xlabel='amount', ylabel='step'>



In [36]:
```python
# Visualization between amount and fraud_transaction_label

plt.figure(figsize=(15,6))
plt.scatter(x='amount',y='fraud_transaction_label',data=Fraud_D)
plt.xlabel('amount')
plt.ylabel('fraud_transaction_label')
```

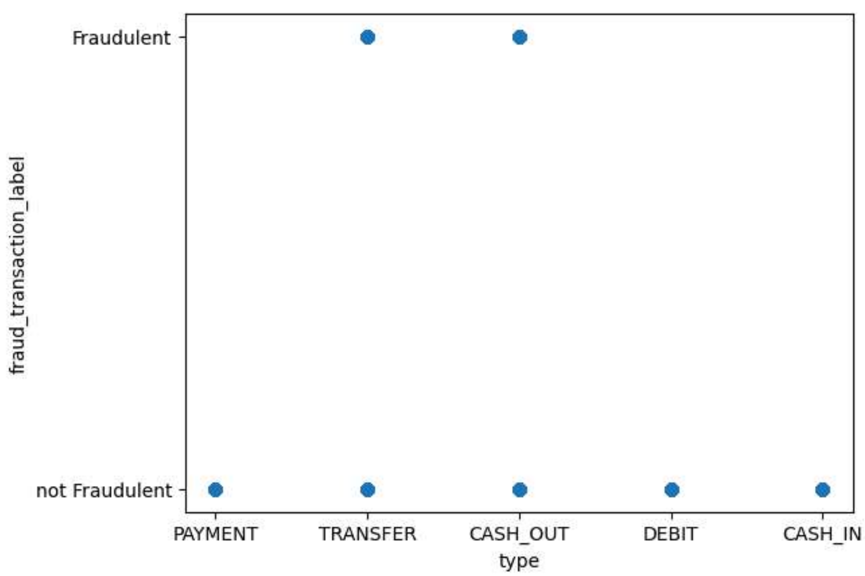
Out[36]:
```
Text(0, 0.5, 'fraud_transaction_label')
```



In [37]:
```python
# Although the amount of fraudulent transactions is very low, majority of them are constituted within 0 and 10,000,000 amou
```

In [38]:
```python
# Visualization between type and isfraud_label

plt.scatter(x='type',y='fraud_transaction_label',data=Fraud_D)
plt.xlabel('type')
plt.ylabel('fraud_transaction_label')
```

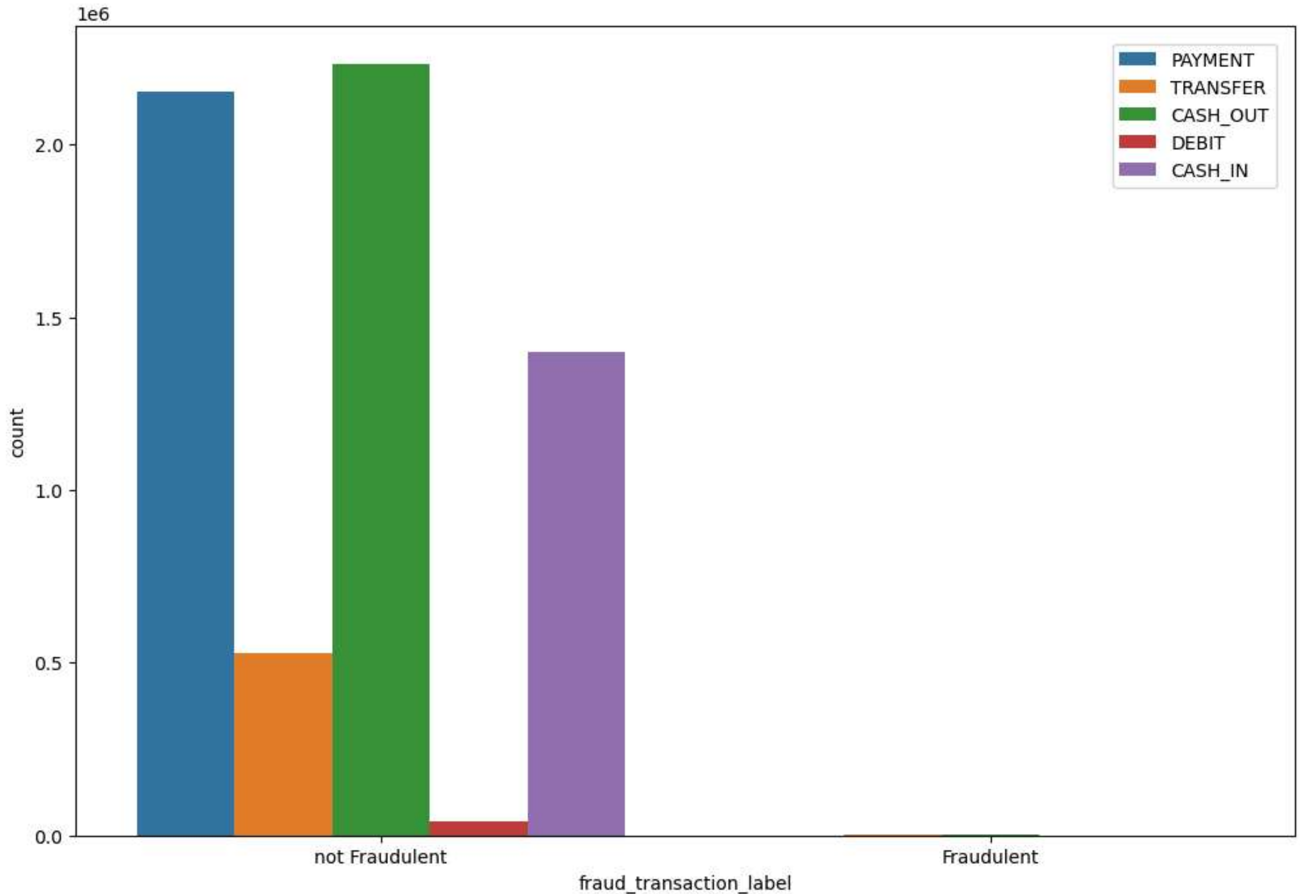
Out[38]:
```
Text(0, 0.5, 'fraud_transaction_label')
```



In [39]:

```python
# Visualization between type and isfraud_label

plt.figure(figsize=(12,8))
sns.countplot(x='fraud_transaction_label',data=Fraud_D,hue='type')
plt.legend(loc=[0.85,0.8])
```

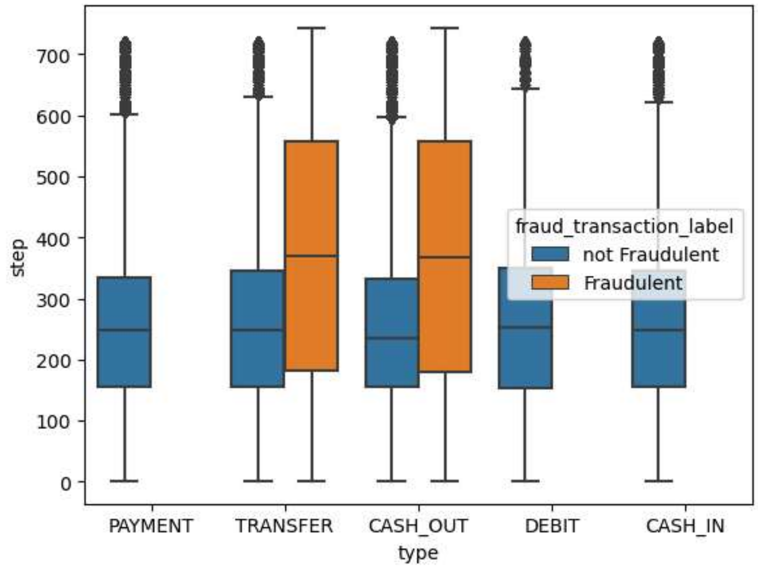
Out[39]: <matplotlib.legend.Legend at 0x1ac381eb850>



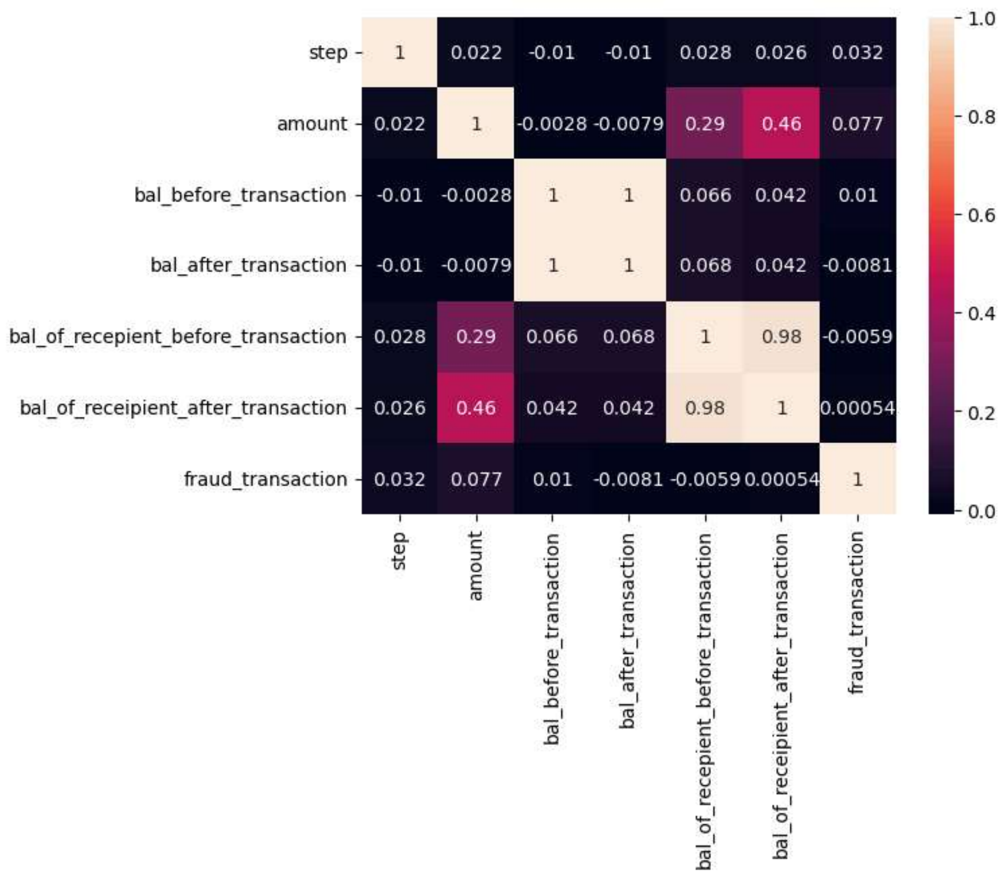
In [40]:

```python
# Both the above graphs indicate that transactions of the type 'transfer' and 'cash out' comprise fraudulent transactions
```

In [41]:
```python
# Multivariate Analysis

# Visualizing btw step,type and isFraud_label

sns.boxplot(x= "type", y= "step", hue ="fraud_transaction_label", data= Fraud_D)
```
Out[41]: <Axes: xlabel='type', ylabel='step'>



In [43]:
```python
# Correlation

corel= Fraud_D.corr()
sns.heatmap(corel, annot =True)
```
Out[43]: <Axes: >



	step	amount	bal_before_transaction	bal_after_transaction	bal_of_recepient_before_transaction	bal_of_receipient_after_transaction	fraud_transaction
step	1	0.022	-0.01	-0.01	0.028	0.026	0.032
amount	0.022	1	-0.0028	-0.0079	0.29	0.46	0.077
bal_before_transaction	-0.01	-0.0028	1	1	0.066	0.042	0.01
bal_after_transaction	-0.01	-0.0079	1	1	0.068	0.042	-0.0081
bal_of_recepient_before_transaction	0.028	0.29	0.066	0.068	1	0.98	-0.0059
bal_of_receipient_after_transaction	0.026	0.46	0.042	0.042	0.98	1	0.00054
fraud_transaction	0.032	0.077	0.01	-0.0081	-0.0059	0.00054	1

In [44]:
```python
# One Hot Encoding
#1. select categorical variables

categorical = ['type']
```

In [45]:
```python
#2. use pd.get_dummies() for one hot encoding
#replace pass with your code

categories_dummies = pd.get_dummies(Fraud_D[categorical])

#view what you have done
categories_dummies.head()
```

Out[45]:

	type_CASH_IN	type_CASH_OUT	type_DEBIT	type_PAYMENT	type_TRANSFER
0	0	0	0	1	0
1	0	0	0	1	0
2	0	0	0	0	1
3	0	1	0	0	0
4	0	0	0	1	0

In [46]:
```python
#join the encoded variables back to the main dataframe using pd.concat()
#pass both data and categories_dummies as a list of their names
#pop out documentation for pd.concat() to clarify

Fraud_D = pd.concat([Fraud_D,categories_dummies], axis=1)

#check what you have done
print(Fraud_D.shape)
Fraud_D.head()
```

```
(6362620, 16)
```

Out[46]:

	step	type	amount	customer_starting_transaction	bal_before_transaction	bal_after_transaction	recipient_of_transaction	bal_of_recepient_before_trar
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	

In [47]:
```python

#remove the initial categorical columns now that we have encoded them
#use the list called categorical to delete all the initially selected columns at once

Fraud_D.drop(categorical, axis = 1, inplace = True)

Fraud_D.drop(columns=['fraud_transaction_label', 'customer_starting_transaction', 'recipient_of_transaction'], inplace=True
```

In [48]:
```python
Fraud_D.head()
```

Out[48]:

	step	amount	bal_before_transaction	bal_after_transaction	bal_of_recepient_before_transaction	bal_of_receipient_after_transaction	fraud_transaction	type_
0	1	9839.64	170136.0	160296.36	0.0	0.0	0	
1	1	1864.28	21249.0	19384.72	0.0	0.0	0	
2	1	181.00	181.0	0.00	0.0	0.0	1	
3	1	181.00	181.0	0.00	21182.0	0.0	1	
4	1	11668.14	41554.0	29885.86	0.0	0.0	0	

In [49]:
```python
# Model Selection, Training and Validation
# Select Target

y = Fraud_D.fraud_transaction
```

In [50]:
```python
X = Fraud_D.drop(['fraud_transaction'], axis = 1)    #Selecting Features
```

In [51]:
```
X
```

Out[51]:

	step	amount	bal_before_transaction	bal_after_transaction	bal_of_recepient_before_transaction	bal_of_receipient_after_transaction	type_CASH_IN
0	1	9839.64	170136.00	160296.36	0.00	0.00	0
1	1	1864.28	21249.00	19384.72	0.00	0.00	0
2	1	181.00	181.00	0.00	0.00	0.00	0
3	1	181.00	181.00	0.00	21182.00	0.00	0
4	1	11668.14	41554.00	29885.86	0.00	0.00	0
...	...	...	...	...	...	...	...
6362615	743	339682.13	339682.13	0.00	0.00	339682.13	0
6362616	743	6311409.28	6311409.28	0.00	0.00	0.00	0
6362617	743	6311409.28	6311409.28	0.00	68488.84	6379898.11	0
6362618	743	850002.52	850002.52	0.00	0.00	0.00	0
6362619	743	850002.52	850002.52	0.00	6510099.11	7360101.63	0

6362620 rows × 11 columns

In [52]:
```
# Import ML Algorithms and Implement Them

#import the libraries we will need
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [53]:
```
## Train test split( training on 80% while testing is 20%)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

In [54]:
```
# Initialize each models
LR = LogisticRegression(random_state=42)
KN = KNeighborsClassifier()
DC = DecisionTreeClassifier(random_state=42)
RF = RandomForestClassifier(random_state=42)
```

In [55]:
```
#create list of your model names
models = [LR,KN,DC,RF]
```

In [56]:
```
def plot_confusion_matrix(y_test,prediction):
    cm_ = confusion_matrix(y_test,prediction)
    plt.figure(figsize = (6,4))
    sns.heatmap(cm_, cmap ='coolwarm', linecolor = 'white', linewidths = 1, annot = True, fmt = 'd')
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()
```

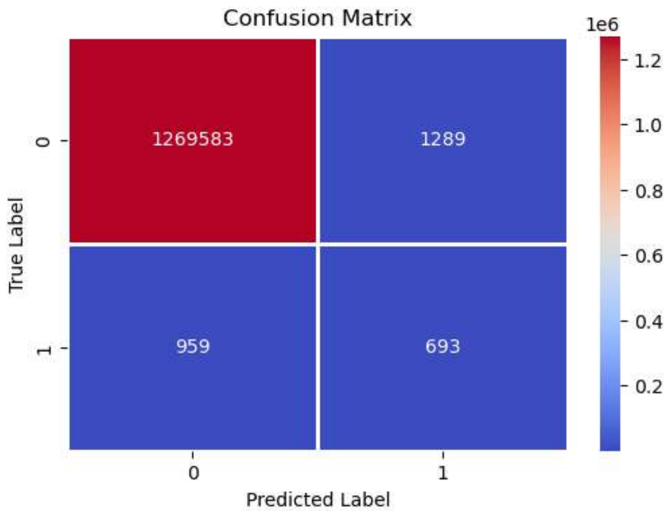
In [57]:
```
from sklearn.metrics import confusion_matrix
```

In [62]:
```
#create function to train a model and evaluate accuracy
def trainer(model,X_train,y_train,X_test,y_test):
    #fit your model
    model.fit(X_train,y_train)
    #predict on the fitted model
    prediction = model.predict(X_test)
    #print evaluation metric
    print('\nFor {}, Accuracy score is {} \n'.format(model.__class__.__name__,accuracy_score(prediction,y_test)))
    print(classification_report(y_test, prediction)) #use this later
    plot_confusion_matrix(y_test,prediction)
```

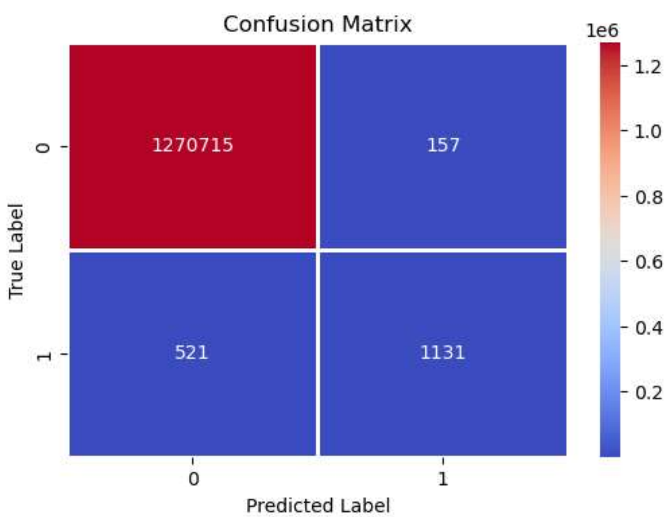
In [63]:

```
#Loop through each model, training in the process
for model in models:
    trainer(model,X_train,y_train,X_test,y_test)
```

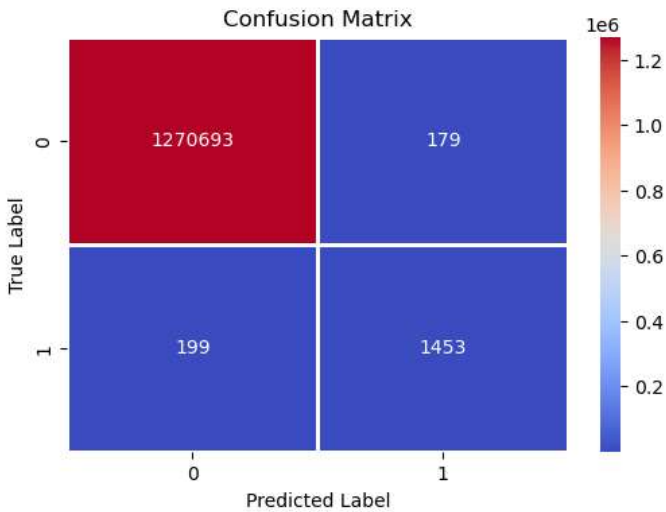
```
For LogisticRegression, Accuracy score is 0.9982334321395903

              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1270872
           1       0.35      0.42      0.38      1652

    accuracy                           1.00   1272524
   macro avg       0.67      0.71      0.69   1272524
weighted avg       1.00      1.00      1.00   1272524
```



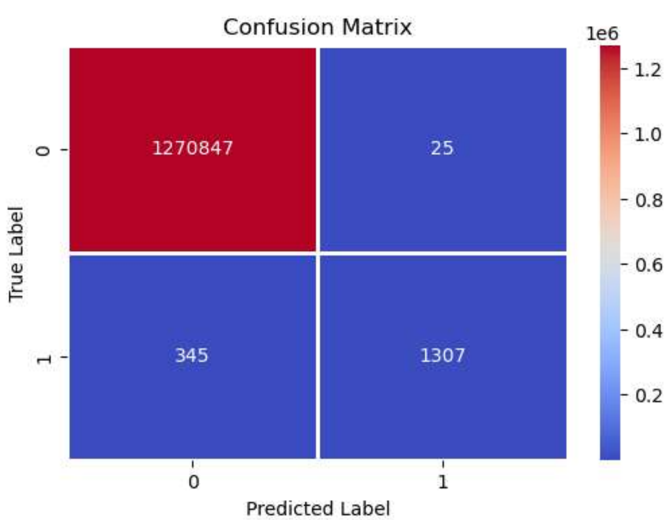

```
For KNeighborsClassifier, Accuracy score is 0.999467200618613

              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1270872
           1       0.88      0.68      0.77      1652

    accuracy                           1.00   1272524
   macro avg       0.94      0.84      0.88   1272524
weighted avg       1.00      1.00      1.00   1272524
```



```
For DecisionTreeClassifier, Accuracy score is 0.9997029525572798

              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1270872
           1       0.89      0.88      0.88      1652

    accuracy                           1.00   1272524
   macro avg       0.95      0.94      0.94   1272524
weighted avg       1.00      1.00      1.00   1272524
```




```
For RandomForestClassifier, Accuracy score is 0.9997092392756443

              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1270872
           1       0.98      0.79      0.88      1652

    accuracy                           1.00   1272524
   macro avg       0.99      0.90      0.94   1272524
weighted avg       1.00      1.00      1.00   1272524
```




```
# Interpretation of the result
# The Decision Tree model with default parameters yields 99.96% accuracy on training data.
# Precision Score: This means that 82% of all the things we predicted came true. that is 82% of clients transactions was detected to be a fraudulent transaction.

# Recall Score: In all the actual positives, we only predicted 82% of it to be true.

# Random Forest Tree model with default parameters yields 99.97% accuracy on training data.
# Precision Score: This means that 99% of all the things we predicted came true. that is 99% of clients transactions was detected to be a fraudulent transaction.

```

```
# Recall Score: In all the actual positives, we only predicted 81% of it to be true.

# Both the Decision Tree and Random Forest models outperform the Logistic Regression and K-Nearest Neighbors model by a wide margin. Since they both have similar recall scores, we should perform a cross-validation of the two models so we may declare which is the best performer with more certainty.
```

In [*]:

```python
# Cross Validation

# Importing the library to perform cross-validation
from sklearn.model_selection import cross_validate

# Running the cross-validation on both Decision Tree and Random Forest models; specifying recall as the scoring metric
DC_scores = cross_validate(DC, X_test, y_test, scoring='recall_macro')
RF_scores = cross_validate(RF, X_test, y_test, scoring='recall_macro')

# Printing the means of the cross-validations for both models
print('Decision Tree Recall Cross-Validation:', np.mean(DC_scores['test_score']))
print('Random Forest Recall Cross-Validation:', np.mean(RF_scores['test_score']))
```

```
# Conclusion
# Upon training and evaluating our classification model, we found that the Random Forest model performed the best by a narrow margin.

# Therefore, Random Forest performs best with recall cross-validation accuracy of 87% which is important for our problem statement where false negative is our priority

# Recommendation
# Transaction History and Frequency - if unaccounted transactions occurs frequently we should confirm genuinity of the transaction with the customer

# Repeated wrong PIN or Password - We should halt the transaction and alert the customer immediately.

# Make customers to change PIN or password often

# Instruct user to use own mobile or computers while doing transactions to avoid phishing attacks

# Increased cybersecurity for banking websites and mobile applications

# Two factor authentication for transaction

# Ensure that blossom bank hire a data engineer that will ensure the dataset is accurate, balanced for proper EDA as there are too many outliers in this data set. This will enable the business to build machime learning models that predict outcomes more accurately with better performance.
```